Introducing Earthly Cloud. Consistent, repeatable builds. Advanced caching for speed. Works with any CI. Get 6,000 build min/mth free! Learn more.

EARTHLY
Super Simple Builds

Golang SQLite `database/sql`

50 minute read Updated: September 19, 2023

Adam Gordon Bell

In this Series

Table of Contents

This article explains how to integrate SQLite with Golang. Earthly simplifies the automation and containerization of Golang SQLite builds. Learn more about Earthly.

Welcome back. I'm an experienced developer, learning Golang by building an activity tracker. Last time I made a command-line client to connect to the JSON Service, but today is all about database persistence using `database/sql`.

If you're curious about the basics of storing persistent data into a SQL database using Golang, this tutorial will be helpful for you. I'm going to be using `sqlite3`, but I'll add lots of headings, so you can skip ahead if `sqlite` is not your thing.

My plan is to add SQLite persistence to the backend service so that my workouts aren't lost if the service goes down. And once I have that, I'll add the `--list` command to my command line client and add an end point for it. it's the type of feature that is simple to do with a SQL backend.

Install SQLite

The first thing I need is to set up my dev environment. I need to install SQLite3 and SQLite-utils:

```
brew install sqlite3
brew install sqlite-utils
```

I'm on Mac OS, but you can find these in your package manager of choice.

Fun Tool: `sqlite-utils`

`sqlite-utils` is a handy tool for working with SQLite databases at the command line. It makes it simple to query for results or insert records from your terminal.

In particular `sqlite-utils` is good at is creating a database schema based on a CSV or JSON schema. So if I start up the service and get a sample JSON doc:

```
> curl -X GET -s localhost:8080 -d '{"id": 1}' | jq '.activity'
{
  "time": "2021-12-09T16:34:04Z",
  "description": "bike class",
  "id": 1
}
```

Then I can use `sqlite-utils` to create a database and table based on this JSON document's structure:

```
$ curl -X GET -s localhost:8080 -d '{"id": 1}' | \
 jq '.activity'  | \
 sqlite-utils insert activities.db activities  -
```

That gives me a good starting point for creating my table – I can never remember the `CREATE TABLE` syntax – and I can use `sqlite-utils` to return the schema it created:

```
> sqlite-utils schema activities.db
```

Output

```
CREATE TABLE [activities] (
   [id] INTEGER
   [time] TEXT,
```

```
    [description] TEXT,
  );
```

I'm going to create this sqlite3 schema manually, but it's helpful to know you can use a tool to create a schema for you.

SQLite3 Create Database

SQLite databases are stored in files with the `.db` extension. I can create one with the schema I want using the sqlite3 command line tool like this:

```
sqlite3 activities.db
```

sqlite3 Copy

```
SQLite version 3.32.3 2020-06-18 14:16:19
Enter ".help" for usage hints.
sqlite> sqlite> CREATE TABLE [activities] (
   ...> id INTEGER NOT NULL PRIMARY KEY,
   ...> time DATETIME NOT NULL,
   ...> description TEXT
   ...> );
```

SQLite, Data Types, and `database/sql`

You may notice that I'm storing time as `DATETIME` whereas sqlite-utils suggested `TEXT` for that column. SQLite is an amazing database but it has an unusual stance on types: it doesn't really care about static types. Richard Hipp, the creator, doesn't even like the term static types. He prefers to call them rigid types (which he thinks are often a mistake.[1])

Because of this stance, there is no statically verified `TIME` or `DATETIME` type in SQLite. Only `INTEGER`, `REAL`, `TEXT`, and `BLOB`. If you set the type as `DATETIME`, you can insert anything you want into it because it's stored as TEXT on disk:

sqlite3 Copy

```
sqlite> insert into activities values
 ...> (NULL,"not a date","christmas eve bike class");
```

```
sqlite> select * from activities;
1|not a date|bike class
```

Why then am I using `DATETIME`? Well, It's helpful to document the type of the field, and also, I'm going to be using `database/sql` in my service, and its scan function may use the column types when converting row values.

Populating the SQLite Database

I'm going to add some sample data to the database.

sqlite3 Copy

```
sqlite> insert into activities values
(NULL,"2021-12-09T16:34:04Z","christmas eve bike class");
sqlite> insert into activities values
(NULL,"2021-12-09T16:56:12Z","cross country skiing is horrible and cold");
sqlite> insert into activities values
(NULL,"2021-12-09T16:56:23Z","sledding with nephew");
```

I can see the data like this:

sqlite3 Copy

```
sqlite> select * from activities;
1|2021-12-09T16:34:04Z|christmas eve bike class
2|2021-12-09T16:56:12Z|cross country skiing is horrible and cold
3|2021-12-09T16:56:23Z|sledding with nephew
```

Commands in `sqlite3` start with a dot `.` so I exit like this:

sqlite3 Copy

```
sqlite> .exit
```

By the way, I rarely select data using the sqlite3 client. Instead, I like to use `sqlite-utils` which has a nice table output view:

```
sqlite-utils activities.db "select * from activities" --table
  id  time                  description
----  --------------------  ----------------------------------------
   1  2021-12-09T16:34:04Z  christmas eve bike class
   2  2021-12-09T16:56:12Z  cross country skiing is horrible and cold
   3  2021-12-09T16:56:23Z  sledding with nephew
```

You can also set **`.mode box`** in your **`.sqliterc`** to get a nicer output out of **`sqlite3`**.

`sqlite-utils` also has a dump command, which is helpful if I want a text backup of my database contents to version control.

```
> sqlite-utils dump activities.db
BEGIN TRANSACTION;
CREATE TABLE [activities] (
id INTEGER NOT NULL PRIMARY KEY,
time TEXT,
description TEXT
);
INSERT INTO "activities" VALUES(1,'2021-12-09T16:34:04Z','christmas eve bi
INSERT INTO "activities" VALUES(2,'2021-12-09T16:56:12Z','cross country sk
INSERT INTO "activities" VALUES(3,'2021-12-09T16:56:23Z','sledding with ne
COMMIT;
```

With my database in place, I can now start in on changes to the service.

Golang SQLite Setup

To use sqlite3 from Golang, I need a database driver. I'm going to use **`go-sqlite3`** which I can install like this:

```
go get github.com/mattn/go-sqlite3
```

*Installing go-sqlite3 requires **`gcc`** and **`CGO_ENABLED=1`***

Finally, let's jump into the Golang code.